IN THE SPECIFICATION

Please amend paragraph [0036] of the originally filed specification as follows:

~~Figure 11~~ Figures 11A and 11B show ~~shows~~ details within the Monitor module and their calling functions between the sub-modules;

Please amend paragraph [00101] of the originally filed specification as follows:

~~Figure 11 shows~~ Figures 11A and 11B show the structural details of the Monitor module 1006, including the various software sub-modules, and the calling functions between the sub-modules of the Monitor module 1006. The Monitor module 1006 includes a Common module 1101 that contains classes used by many modules, a MonitorManager module 1102 that manages the other sub-modules (including the DeviceODBC module 1104, the Device module 1110, and the HWaccess module 1116) to complete the tasks defined by interface functions as illustrated in FIG. 10. Specifically, the DeviceODBC module 1104 is accessed in order to access external device information through the standard interface. The HWaccess module 1116 obtains vendor, model, unique ID, and status information from the monitored devices using a selected communication protocol from among a plurality of communication protocols (e.g., HTTP, SNMP, and FTP). Each of the Monitor software modules will be described in more detail below.

Please amend paragraph [00123] of the originally filed specification as follows:

Figure 12 shows the data structure used by the HWaccess module 1116, as illustrated in ~~Figure. 11~~ Figures 11A and 11B, to exchange information for retrieval of values associated with key values received by the HWaccess module 1116. For example, the SKeyValueInfo data structure, as shown in Figure 12, is used to determine how to obtain information corresponding to a particular information type (corresponding to m_infoType 1202) within a

given web page. Typically, a multitude of vendors use vendor-specific identifiers and nomenclature to identify key information, displayed on their respective web pages, related to a monitored device. For example, to determine the number of pages printed by a printer device, Hewlett Packard uses the "Page Count" feature, while Xerox identifies the same using a "Total Sheet Delivered" feature. A feature of the present invention is to overcome the vendor-to-vendor variances and thereby provide a standardized and uniform method of identifying device-specific information and extract the value corresponding to the information by using a data structure/SKeyValueInfo structure 1200. The SKeyValueInfo data structure 1200 includes attributes that are public.

Please amend paragraph [00129] of the originally filed specification as follows:

Referring to Figure 15, there is shown a vector 1500 having reference to the devices created by the CDeviceFactory 1106 and used by the MonitorManager 1102, as illustrated in Figures 13 and 14. MonitorManager 1102 stores device pointers, such as for example, Pointer to CDevice Object 1502, ~~and~~ Pointer to CDevice Object 1504, CDevice Object 1506, and CDevice object 1508 created by CDeviceFactory 1106, in the vector. The vector sequence is iterated to obtain the status of a monitored device. Polling of monitored devices is performed over the device object by issuing an obtainStatus command. Once the status of each of the software objects is obtained, such status is updated through the DeviceODBC 1104. The status monitor sequence was described above at Figure 14, and will not be repeated herein.

Please amend paragraph [00141] of the originally filed specification as follows:

Figure 16 illustrates the DeviceODBC module class structure according to the present invention, and shows how the CAbsProtocolParameters class structure is used within the

DeviceODBC module. The CAbsProtocolParameters class 1600 is designed to interface with the monitor database 1014 and to obtain information for accessing the monitored devices using a particular communication protocol. The CAbsProtocolParameters class has two virtual functions which are protocol-independent:

(1) std::string obtainProtocolName(void); and

(2) bool obtainParameterVector(std::vector<SParameter> & out_ParameterVector, const std::string in_sIP).

Using these functions, the CDeviceODBC class 1602 can handle as many protocols and their associated parameter names and values through the pointer to the CAbsProtocolParameter type, without identifying the protocol. The obtained information for each device (e.g., IP Address) is stored in the data structure of Figure 18 and passed to the MonitorManager module 1102 through the obtainConfig function. From the CDeviceODBC perspective, all the objects used to obtain the protocol name and the associated parameter names and values are considered to be a type of CAbsProtocol Parameters. When a new parameter is added, therefore, the new object should be created and stored in the vector of pointers to CAbsProtocolParameters class. The other functions do not need to be changed. The DeviceODBC module class structure also includes CDeviceInfoData 1602, CDeviceHistoryData 1606, CSNMPProtocolParameters 1608, CHTTPProtocolParamaters 1610, CFTPProtocolParamaters 1612, CDeviceInfoTable 1614, CDeviceHistoryTable 1616, CSNMPTable 1618, CHTTPTable 1620, CFTPTable 1622, and CRecordSet 1624.

Please amend paragraph [00135] of the originally filed specification as follows:

Figures 21 and 22 illustrate the data structures included in the HTTP and FTP portions of the support database 1024 and includes data structures analogous to the data structures described above with regard to Figure 20. Figure 21 shows that the data structures included in

the HTTP portions of the support database 1024 include HTTPVendorModel 2100, HTTPSupportedVendorModelDelay 2102, HTTPUniqueIdWebPage2104, HTTPVendorModelWebPage 2106, EnumCorrespondence 2108, and HTTPStatusKeyValue 2110. Figure 22 shows that the data structures included in the FTP portions of the support database 1024 include FTPVendor 2200, FTPVendorModelDirectFileID 2202, FTPStatus 2204, and EnumCorrespondence 2206.

Please amend paragraph [00142] of the originally filed specification as follows:

Figure 23 illustrates the HWaccess module class structure according to the present invention, and shows how the CAbsProtocol class structure 2308 is used within the HWaccess module. The HWaccess module class includes CHWaccess class 2300. The CAbsProtocol class 2308 is designed to interface with the support database 1024 and to obtain support information for extracting status information from the monitored devices using a particular communication protocol (i.e., HTTP 2302, SNMP 2304, or FTP 2306). The CAbsProtocol class 2308 has the virtual functions described above: initBegin, initEnd, canAccessIP, obtainVendor, obtainModel, obtainUniqueID, obtainEventStatus, and obtainStatus. However, the first parameter of the functions obtainEventStatus and obtainStatus have a different type. Instead of std::map<infoType, std::string>, the status type is std::map<infoType, std::pair<std:;string, int>>. This allows for the accommodation of *nRelativePriority*, which allows the highest priority status information to be obtained, as discussed above. For each device and each protocol, the system can check if the particular infoType that can be obtained for the device is already in the status map data . If it is not in the status map data, the particular infoType is added to the information to be extracted for the protocol. If it is in the status map data, the nRelativePriority is compared. If the priority in the status map data is higher or the same, the infoType is not to be obtained using the

protocol. If the priority in the status map data is lower, the intoType is added to the information to be extracted and replaced in the status map data. The HW access module class structure also includes CRecordSet 2310.

Please amend paragraph [00144] of the originally filed specification as follows:

Figure 24 illustrates the uniform class structure within the HTTP, SNMP, and FTP modules shown in Figure 23 and the corresponding relationship to the CAbsProtocol class. Note that "XXX" refers to a particular protocol (e.g., HTTP, SNMP, or FTP), and that the present invention is designed for an expansion of the number of protocols. XXXP 2400 is used to generically represent HTTP, SNMP, or FTP. This use of CAbsProtocol 2402 allows the CHWaccess class to handle the future protocol support without changing the main functions. When a new protocol is added to be supported, the new protocol should follow the structure of Figures 23 and 24. Figure 24 shows CXXXProtocol 2404, XXXP::XXXPaccess 2406, XXXP::XXXPODBC 2408, and CRecordSet 2410 to generally represent a new protocol to be added. Then, the CHWaccess class only needs to create the added protocol object and put it in the vector of the CAbsProtocol object pointers.